Mustererkennung - Aufgabenblatt 03

André Hacker und Dimitri Schachmann

1. Gaussverteilung

Zuerst die Lösung nach der neuen Definition der Aufgabe (gemäß Mail):
Der Schwellenwert $S(x)$ ist eine Funktion in Abhängigkeit vom zu klassifizierenden Objekt x (hier 1-d)

$$S(x) = P(C_1|x) - P(C_2|x)$$

Ist $S(x)$ positiv, ist die Wahrscheinlichkeit für Klasse C_1 höher und wir wählen diese. Wir formen mit Hilfe Bayes-rule um

$$\begin{aligned} S(x) &= P(C_1|x) - P(C_2|x) \\ &= \frac{P(C_1 \cap x)}{P(x)} - \frac{P(C_2 \cap x)}{P(x)} \\ &= \frac{P(C_1)P(x|C_1)}{P(x)} - \frac{P(C_2)P(x|C_1)}{P(x)} \\ &= \frac{P(C_1)P(x|C_1) - P(C_2)P(x|C_1)}{P(x)} \end{aligned}$$

Die einzelnen Komponenten haben übrigens spezielle Bezeichnungen:

$$Posterior = P(C_1|x) = \frac{P(C_1)P(x|C_1)}{P(x)} = \frac{Prior \cdot Posterior}{evidence}$$

Da uns nur interessiert ob $S(x)$ größer oder kleiner 0 ist können wir $P(x)$ ignorieren (ist der gleiche Teiler für beide Brüche, wir können also die Gleichung mit $P(x)$ multiplizieren). Die Wahrscheinlichkeit $P(x|C_1)$ ist nun genau durch die Dichtefunktion der Normalverteilung für C_1 beschrieben, so dass sich für die Klassifizierung folgende Formel ergibt

$$\begin{aligned} S'(x) &= P(C_1)P(x|C_1) - P(C_2)P(x|C_1) \\ &= P(C_1) \cdot \frac{1}{\sqrt{2\pi}\sigma_1} e^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}} - P(C_2) \cdot \frac{1}{\sqrt{2\pi}\sigma_2} e^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}} \end{aligned}$$

1. Gaussverteilung (alte Definition)

Nun die Lösung für die Aufgabe wie sie im Tutorium besprochen und auf dem Blatt beschrieben war (hatten wir schon fertig als die Mail kam)
Gegeben 2 eindimensionale Normalverteilungen $f_1(x, \mu_1, \sigma_1)$, $f_2(x, \mu_2, \sigma_2)$ kann man den Schwellenwert für x berechnen indem man die folgende Gleichung nach x auflöst:

$$f_1(x, \mu_1, \sigma_1) = f_2(x, \mu_2, \sigma_2)$$

$$\frac{1}{\sqrt{2\pi}\sigma_1} e^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}} = \frac{1}{\sqrt{2\pi}\sigma_2} e^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}}$$

Der Schwellenwert x ist dann der Schnittpunkt der beiden Kurven.
Die Klassifizierung ist auf den ersten Blick denkbar einfach: Liegt ein Punkt x' vom Schwellwert aus gesehen eher bei μ_1, so gehört er zu Klasse 1, und umgekehrt.
Das ist allerdings zu kurz gedacht, weil tatsächlich gibt es zwei Schnittpunkte. Z.B. gibt es nicht nur einen Schwellenwert, wenn die Verteilungen dich beieinander liegen aber sehr unterschiedliche Höhe (Varinanz) haben.

Durch einsetzen und p-q-Formel haben wir folgendes erhalten (nicht mehr abgetippt, da nach neuer Aufgabenstellung nicht gefordert):

$$x^2 + x\left(\frac{2(\mu_2\sigma_1^2 - \mu_1\sigma_2^2)}{\sigma_2^2 - \sigma_1^2}\right) + \left(\frac{\mu_1^2\sigma_2^2 - \mu_2^2\sigma_1^2 - 2ln(\frac{\sigma_2}{\sigma_1})\sigma_1^2\sigma_2^2}{\sigma_2^2 - \sigma_1^2}\right) = 0$$

$$x = \frac{2(\mu_1\sigma_2^2 - \mu_2\sigma_1^2)}{\sigma_2^2 - \sigma_1^2} \pm \sqrt{\left(\frac{2(\mu_2\sigma_1^2 - \mu_1\sigma_2^2)}{\sigma_2^2 - \sigma_1^2}\right)^2 - \left(\frac{\mu_1^2\sigma_2^2 - \mu_2^2\sigma_1^2 - 2ln(\frac{\sigma_2}{\sigma_1})\sigma_1^2\sigma_2^2}{\sigma_2^2 - \sigma_1^2}\right)}$$

Zwei Spezialfälle ergeben sich: Wenn $\mu_1 = \mu_2$ gilt, gibt es keinen Schwellenwert
Wenn $\sigma_1 = \sigma_2$ gilt, liegt der Schwellwert genau in der Mitte von μ_1 und μ_2 also $x = \frac{\mu_1 + \mu_2}{2}$

Das ganze haben wir implementiert und mit dem Wert verglichen den Matlab berechnet, wenn es die Gleichung ganz oben auflöst (identisch).

```
function threshold

  %threshold should be 0.25
  t1 = myThreshold (0.5, 1, 0, 1);

  % Let's see what Matlab computes and compare with this:
  t2 = solve( '1 / (sqrt(2*pi) * 1) * exp( -(x-0)^2 / (1^2) ) = 1 / (sqrt(2*pi) * 1) * exp( -(x-0
  %solve( 'x-2 = -x' ) %should meet at x=1

  fprintf('Our result: %d\n', t1);
  fprintf('Matlab result: %d\n', double(t2));

end


function t = myThreshold(m1, s1, m2, s2)

  % Calculate threshold
  % Attention: Not definied for s1=s2. If s1=s2 there is another formula (not implemented here)
  % TODO: Use the simplified formula in this case
  if s1==s2
    s2 = s2 + 0.00001;
  end

  % There is only one point of intersection.
  % Normalize, so m1 is always left from m2
  fact = -1;
```

```
  if m2 > m1
    fact = 1;
  end

  t = 2*(m1*s2^2 - m2*s1^2)/(2*(s2^2 - s1^2));
  t = t + fact ...
    * sqrt( (2*(m2*s1^2 - m1*s2^2)/(2*(s2^2 - s1^2)))^2 - ...
      (m1^2*s2^2 - m2^2*s1^2 - ...
        2*log(s2/s1)*s1^2*s2^2)/(s2^2 - s1^2) );

end
```

2. Fisher's Diskriminante

Unser Algorithmus geht für jede Kombination von 2 Klassen folgendermaßen vor

1. Den Erwartungswert μ_1, μ_2 und Kovarianzmatrizen Σ_1, Σ_2 (16-dimensional) für beide Klassen berechnen
2. Optimale Projektionsrichtung mittels Fisher Kriterium berechnen
3. Daraus ergeben sich die neuen eindimensionalen Klassen, die wir mit μ_1, μ_2 und σ_1, σ_2 beschreiben.
4. Ein neues Element $\vec{x}$ wird nun klassifiziert indem die Projektion gebildet wird und dann aufgrund des Schwellwertes (siehe Aufgabe 1) die Klasse gewählt wird.

Folgende (symmetrische) Matrix enthält die Erfolgsraten in Prozent.

	0	1	2	3	4	5	6	7	8	9
0	0	88.858	97.937	96.71	91.334	93.983	72.818	98.212	48.069	53.076
1	88.858	0	85.44	88.714	98.214	61.373	43.286	50.962	51.143	68.714
2	97.937	85.44	0	99	65.385	100	94	77.61	94.857	99.857
3	96.71	88.714	99	0	87.714	71.237	74.405	46.429	85.268	50
4	91.334	98.214	65.385	87.714	0	89.27	81.429	91.484	63.143	54.286
5	93.983	61.373	100	71.237	89.27	0	87.034	78.827	50.82	54.545
6	72.818	43.286	94	74.405	81.429	87.034	0	55.714	86.905	95.536
7	98.212	50.962	77.61	46.429	91.484	78.827	55.714	0	85.143	62.143
8	48.069	51.143	94.857	85.268	63.143	50.82	86.905	85.143	0	60.714
9	53.076	68.714	99.857	50	54.286	54.545	95.536	62.143	60.714	0

Um unsere Implementierung zu testen haben wir drei einfache Einfache zweidimensionale Beispiele gewählt und die Fisher-Diskriminante berechnet und dargestellt (Linie). Dargestellt werden die Samples der beiden Klassen, die Means (Mu), sowie die Means der Projektion (Mu Projection). Hier die Ergebnisse:

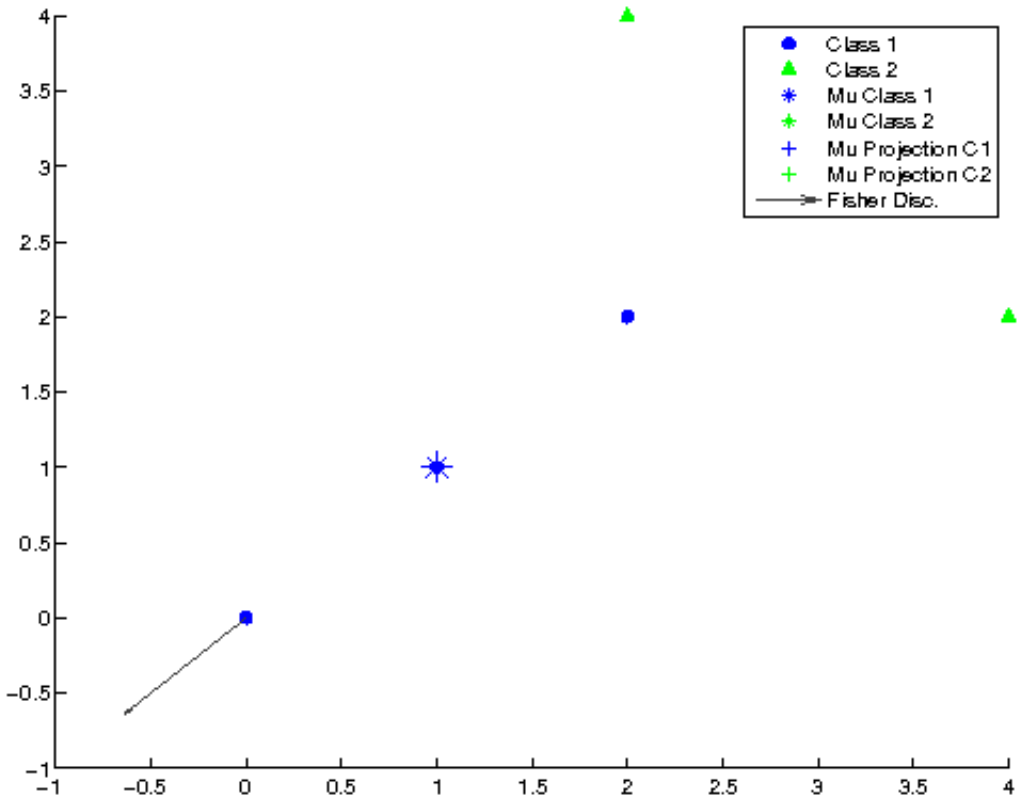

Abbildung 1: Beispiel 2D Fisher Diskriminante

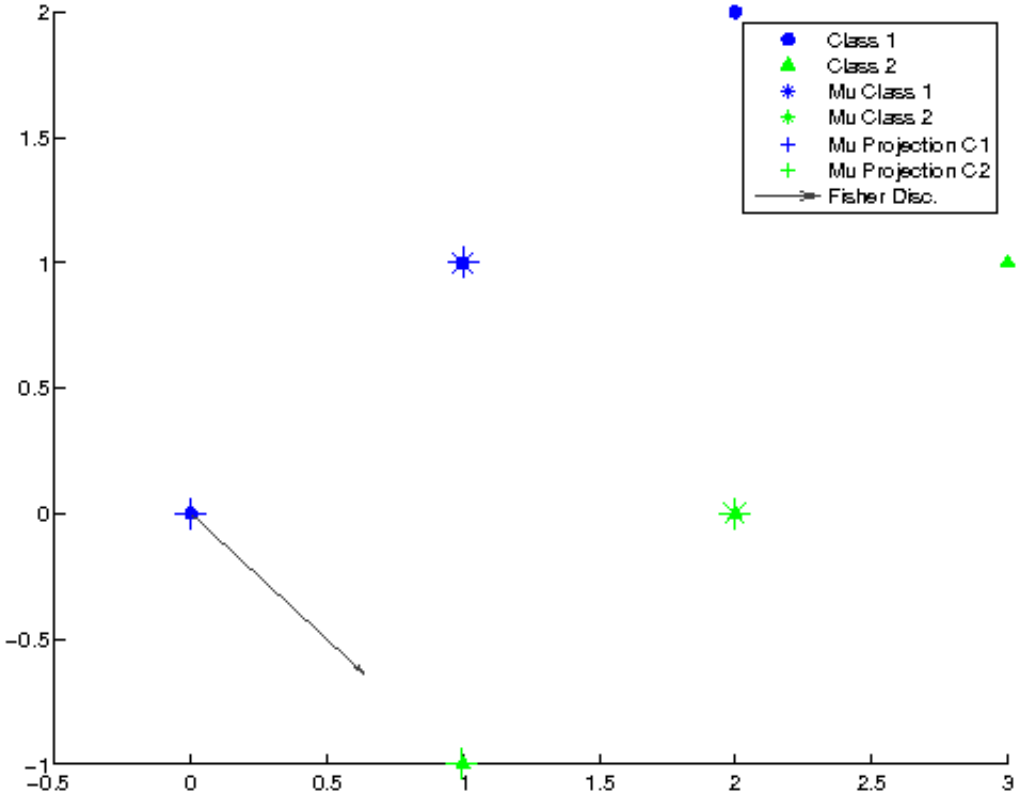

Abbildung 2: Beispiel 2D Fisher Diskriminante

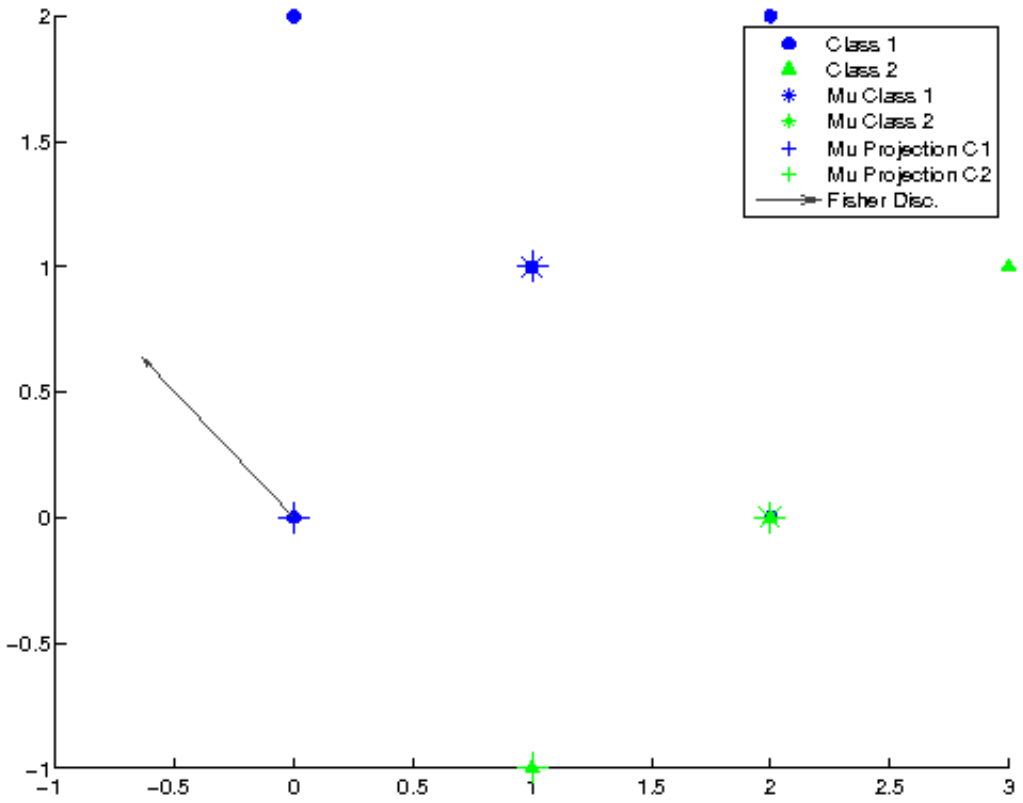

Abbildung 3: Beispiel 2D Fisher Diskriminante

Hier die Implementierung:

```
function fisher

  % Load input into (nx17) Matrix.
  % last column stores the class number
  tra = load('pendigits.tra');
  tes = load('pendigits.tes');

  % Iterate through classes
  result = zeros(10,10);
  for i=0:9
    for j=0:i-1
      samples0 = filterByClass(tra, i);
      samples1 = filterByClass(tra, j);
      test01 = filterByClass(tes, [i j]);
      result(i+1,j+1) = fisher2(samples0, samples1, test01, [i j]);
      result(j+1,i+1) = result(i+1,j+1);
    end
  end
  result = result * 100
```

```
  meanSuccessRate = mean(mean(result))

  dlmwrite('fisher_results.mat', result, ' ');

  % Simple examples that are plotted
  fisher2dTest();

end

% Test-function to generate some simple 2d classification problems
function fisher2dTest

  % Example
  samples1 = [0 0; 1 1; 2 2];
  samples2 = [2 4; 3 3; 4 2];
  fisher2(samples1, samples2, [0 0], [1 2], 'fisher1');

  % Example with s1 = s2
  samples1 = [0 0; 1 1; 2 2];
  samples2 = [1 -1; 2 0; 3 1];
  fisher2(samples1, samples2, [0 0], [1 2], 'fisher2');

  % Example with sigma1 having many zeros
  samples1 = [0 0; 1 1; 2 2; 0 2; 2 0];
  samples2 = [1 -1; 2 0; 3 1];
  fisher2(samples1, samples2, [0 0], [1 2], 'fisher3');

end


% Classification function for two classes
% Computes the fisher discriminant, returns success rate
function success = fisher2(samples1, samples2, testdata, classes, name)

  % Compute mu's and sigma's (covariance matrices)
  mu1 = getMean(samples1(:,1:end-1));
  sigma1 = getCovar(samples1(:,1:end-1), mu1);
  mu2 = getMean(samples2(:,1:end-1));
  sigma2 = getCovar(samples2(:,1:end-1), mu2);

  % Determine optimum projection using fisher criteria
  if sigma1 == sigma2
    % add noise
    sigma2 = 0.01 * rand(size(sigma2,1), size(sigma2,2));
  end
  a = (mu1 - mu2) * (sigma1 - sigma2)^(-1);
  % Normalize to lenght 1
  a = a / sqrt(a*a');

  % Compute mu's and sigma's for projection
  % In script we have column-vectors, here row-vectors
  psigma1 = a * sigma1 * a';
  psigma2 = a * sigma2 * a';
```

```
  pmu1 = project(mu1, a);
  pmu2 = project(mu2, a);

  % --------------
  % CLASSIFY
  % --------------
  success = fisherClassify([pmu1 pmu2], [psigma1 psigma2], a, testdata, classes);

  % Plot 2d example
  if size(a,2)==2
    plotfisher(samples1(:,1:end-1), samples2(:,1:end-1), ...
      mu1, sigma1, mu2, sigma2, ...
      pmu1, psigma1, pmu2, psigma2, a, name);
  end

end


% Classify based on two normal distributions
% uses projection on fisher discriminant
% Returns success rate
function success = fisherClassify(means, sigmas, disc, testdata, classes)
  % Project all to fisher discriminant
  projection = project(testdata(:,1:end-1), disc);

  % Get prob. of each class and projection
  P = [gaussDensity(means(1), sigmas(1), projection) ...
    gaussDensity(means(2), sigmas(2), projection) ];

  % Take class with max. prob.
  % For each row get the index that has the highest value (probability)
  [maxValue maxIndex] = max(P,[],2);

  % Replace index with number(class) it represents
  maxIndex = classes(maxIndex)';

  % Write result (predicted and real class)
  % dlmwrite('recognition_results.mat', [maxIndex testdata(:,end)], ' ');

  success = analyze([maxIndex testdata(:,end)]);

end


function success = analyze(isVsShould)

  hit = 0;
  miss = 0;

  misses = zeros(10,10);
  hits = zeros(1,10);

  for k = 1:size(isVsShould, 1)
```

```
    if isVsShould(k,1) == isVsShould(k,2)
      hit = hit +1;
      hits(isVsShould(k,2)+1) = hits(isVsShould(k,2)+1) + 1;
    else
      miss = miss + 1;
      misses(isVsShould(k,1)+1,isVsShould(k,2)+1) = misses(isVsShould(k,1)+1,isVsShould(k,2)+1) +
    end;
  end;

  success = 1-(miss/(hit+miss));
end


% Return projection from x on a (row vectors)
% x: rows (1 or many) with items to project
% a: row vector where to project on
function p = project(x, a)
  if (size(x,1)>1)
    p = dot(x, repmat(a, size(x,1),1), 2);
  %  p = p * a
  else
    p = dot(x,a);  % Just take the scaling factor
    %p = (dot(x,a)/dot(a,a)) * a;
  end

end

function p = gaussDensity(mu, sigma, data)
  normalize = 1 / (sqrt(2*pi) * sigma);
  p = zeros(size(data,1), 1);
  for i=1:size(data,1)
    p(i) = normalize * exp( -(data(i)-mu)^2 / (sigma^2) );
  end

end

% Input: (nx17)-Matrix with labled samples
% Output: rows with class c (without label column)
function r = filterByClass(samples, c)
  r = samples(ismember(samples(:,end),c),:);
end

% Computes mean based on matrix with observations
% Input: rows with training data for one class
% Output: Mean (row vector)
function m = getMean(data)
  m = 1/size(data,1) * sum(data);
end

% Compute Covariance matrix for 16-d samples
function s = getCovar(samples, mu)
  % Normalize and compute covar
  samples = samples - (ones(size(samples,1),1) * mu);
```

```
  s = samples' * samples;
  s = s / size(samples,1);

  % make some noise;)
  s = s + (0.0001 * eye(size(s)));
end


function plotfisher(samples1, samples2, m1, s1, m2, s2, ...
  pm1, ps1, pm2, ps2, a, name)
  h = figure;
  % http://www.mathworks.de/de/help/matlab/ref/colorspec.html
  % http://www.mathworks.de/de/help/matlab/ref/linespec.html
  scatter(samples1(:,1), samples1(:,2), 'bo', 'filled', 'Displayname', 'Class 1');
  hold on;
  scatter(samples2(:,1), samples2(:,2), 'g^', 'filled', 'Displayname', 'Class 2');
  scatter(m1(1), m1(2), 200, 'b*', 'Displayname', 'Mu Class 1');
  scatter(m2(1), m2(2), 200, 'g*', 'Displayname', 'Mu Class 2');
  scatter(pm1(1), pm1(2), 200, 'b+', 'Displayname', 'Mu Projection C1');
  scatter(pm2(1), pm2(2), 200, 'g+', 'Displayname', 'Mu Projection C2');
  quiver(0,0,a(1),a(2), 'Displayname', 'Fisher Disc.');

  legend('show');

  print(h,'-dpng',[name '.png']);
  %print(h,'-dpng',['fisher-' datestr(now, 'yyyy-mm-dd HH:MM:SS') '.png']);
end
```

3. Partielle Ableitung

a)

1. Plot der Funktion

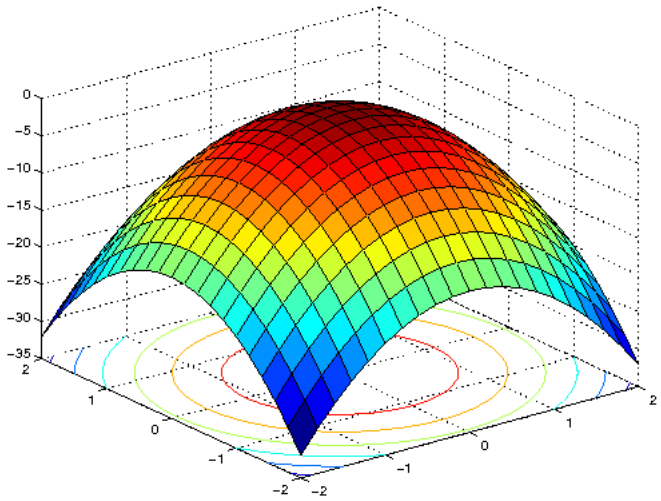

Abbildung 4: Funktionsoberfläche

2. Partielle Ableitungen

$$f(x,y) = -4 \cdot (x^2 + y^2)$$

$$\frac{\partial f}{\partial x} = -8x$$

$$\frac{\partial f}{\partial y} = -8y$$

3. Plots der partiellen Ableitungen

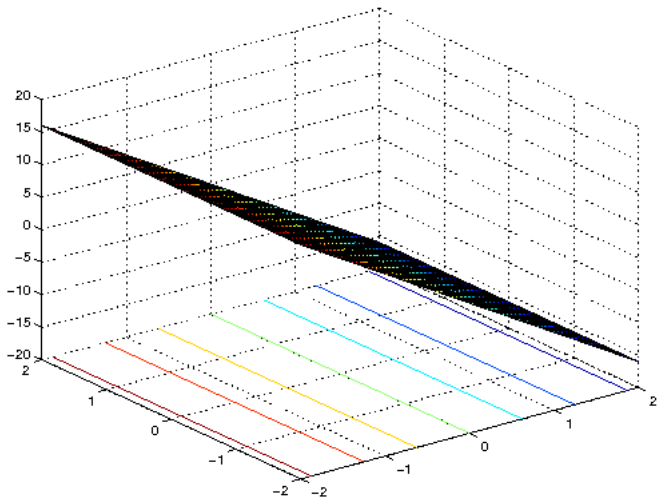

Abbildung 5: Partielle Ableitung nach X

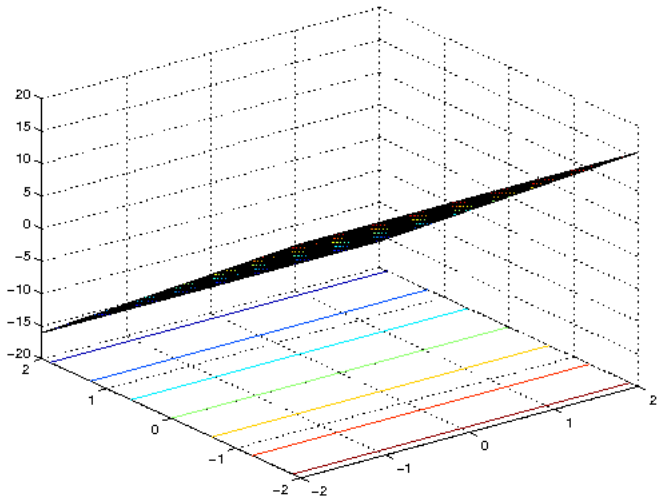

Abbildung 6: Partielle Ableitung nach Y

4. Quiver Plot
Interpretation: siehe Teil b) (analog)

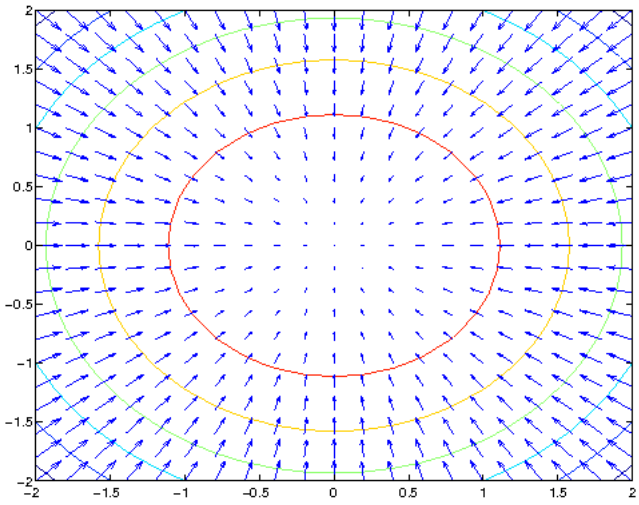

Abbildung 7: Kontur und Quiver Plot

b)

1. Plot der Funktion

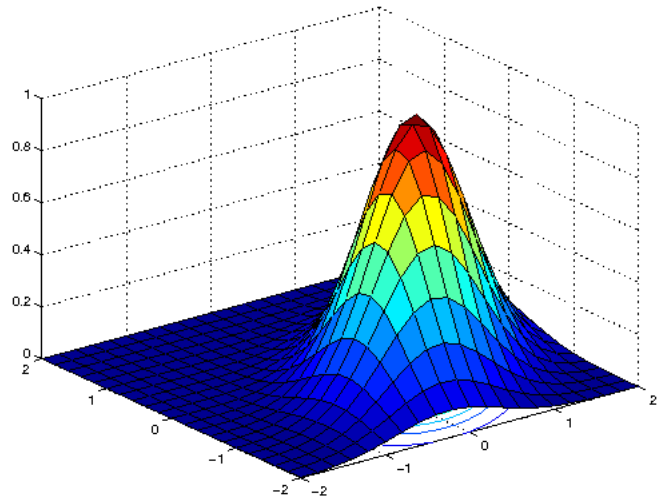

Abbildung 8: Funktionsoberfläche

2. Partielle Ableitungen
Wir formen zuerst die Funktion um (ausmultiplizieren), um besser ableiten zu können (Linearität ausnutzen):

$$\begin{aligned} f(x,y) &= e^{-\frac{(x-1/2)^2-(x-1/2)(y+1/2)+(y+1/2)^2}{3/4}} \\ &= e^{\frac{1}{3}(-4x^2+6x-4y^2-6y+4xy-3)} \end{aligned}$$

Wir haben einen Ausdruck der Form $f(g(x,y))$ mit $f = exp$, also wenden wir die Kettenregel an und leiten partiell ab.

$$\frac{\partial f}{\partial x} = e^{\frac{1}{3}(-4x^2+6x-4y^2-6y+4xy-3)} \cdot \frac{1}{3}(-8x+6+4y)$$

$$\frac{\partial f}{\partial x} = e^{\frac{1}{3}(-4x^2+6x-4y^2-6y+4xy-3)} \cdot \frac{1}{3}(-8y-6+4x)$$

3. Plots der partiellen Ableitungen

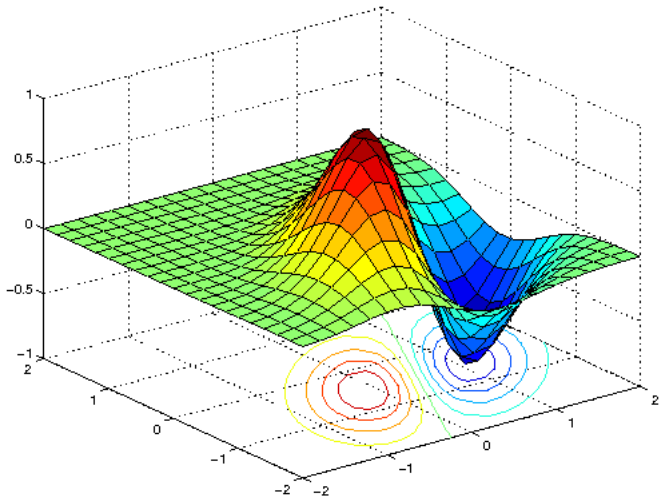

Abbildung 9: Partielle Ableitung nach X

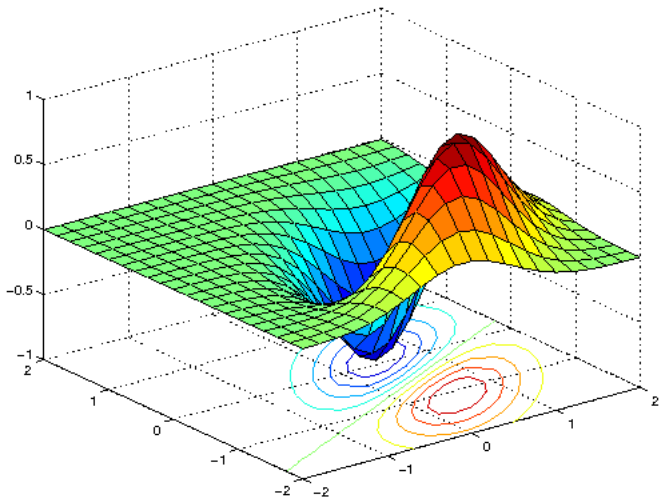

Abbildung 10: Partielle Ableitung nach Y

4. Quiver Plot
Interpretation: Die Vektoren gehen in die Richtung (bezogen auf x und y), in der die Werte der ursprünglichen Funktion am meisten steigen. Daher sind sie auch über die partiellen Ableitungen definiert, weil diese ja gerade die Steigung an der jeweiligen Stelle beschreiben (zumindest jeweils auf eine Komponente bezogen).
Die Länge der Vektoren veranschaulicht, wie stark die Steigung an dieser Stelle ist.
Soweit ich das verstehe entspricht das Quiver-Plot der Darstellung des sogenannten Gradienten, zumindest für ein paar ausgewählte Stellen an denen die Vektoren dann ihren Ursprung haben.

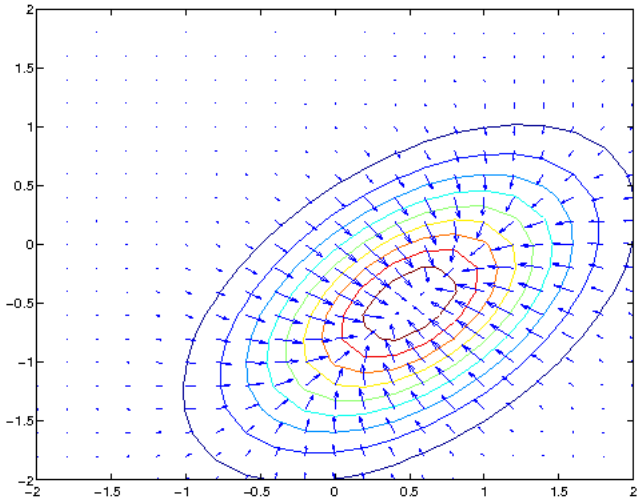

Abbildung 11: Kontur und Quiver Plot

Hier der Source code:

```
function plotderivation()
  [X, Y] = meshgrid(-2:0.2:2);
  plotfigure(X, Y, @f1, 'task3-f1');
  plotfigure(X, Y, @f1derivatedx, 'task3-f1-derivatedX');
  plotfigure(X, Y, @f1derivatedy, 'task3-f1-derivatedY');
  plotwithquiver(X, Y, @f1, @f1derivatedx, @f1derivatedy, 'task3-f1-quiver');

  plotfigure(X, Y, @f2, 'task3-f2');
  plotfigure(X, Y, @f2derivatedx, 'task3-f2-derivatedX');
  plotfigure(X, Y, @f2derivatedy, 'task3-f2-derivatedY');
  plotwithquiver(X, Y, @f2, @f2derivatedx, @f2derivatedy, 'task3-f2-quiver');
end

function plotfigure(X, Y, fun, name)
  Z = fun(X,Y);
  h = figure;
  surfc(X, Y, Z);
  print(h,'-dpng',[name '.png']);
end

function plotwithquiver(X, Y, fun, derix, deriy, name)
  Z = fun(X,Y);
  DX = derix(X,Y);  % same as gradient(Z)!
  DY = deriy(X,Y);
  h = figure;
  contour(X, Y, Z);
  hold on;
  quiver(X, Y, DX, DY);
  print(h,'-dpng',[name '.png']);
end
```

```
% --------------------
% FUNCTIONS TO PLOT
% --------------------
function r = f1(x, y)
  r = -4*(x.^2+y.^2);
end

function r = f1derivatedx(x, y)
  r = -8*x;
end

function r = f1derivatedy(x, y)
  r = -8*y;
end

function r = f2(x, y)
  r = exp(1/3*( -4*x.^2 + 6*x -4*y.^2 - 6*y + 4*x.*y - 3 ));
end

function r = f2derivatedx(x, y)
  r = exp(1/3 *( -4*x.^2 + 6*x -4*y.^2 - 6*y + 4*x.*y - 3 ));
  r = r * 1/3 .* (-8*x + 4*y + 6);
end

function r = f2derivatedy(x, y)
  r = exp(1/3*( -4*x.^2 + 6*x -4*y.^2 - 6*y + 4*x.*y - 3 ));
  r = r * 1/3 .* (-8*y + 4*x - 6);
end


% Links & Comments
% http://www.mathworks.de/de/help/matlab/visualize/representing-a-matrix-as-a-surface.html
% Surface plots are useful for visualizing matrices that are too large to display in numerical fo
%
% Visualizing functions of 2 variables:
% http://www.mathworks.de/de/help/matlab/visualize/representing-a-matrix-as-a-surface.html#f0-520
```